Terraform
自动化管理云基础设施

徐小东

Terraform：自动化管理云基础设施

徐小东

献给

海燕和铭基

目录

表格

插图

更新

你可以从 https://selfhostedserver.com/terraform 获取本书的更新版本，包括 PDF、ePub 和 Mobi 格式。

- 2019.09.27：初版

作者简介

徐小东，网名 toy，GNU/Linux 爱好者，DevOps 践行者。喜技术，好分享，通过 https://linuxtoy.org 网站数年间原创及翻译文章达 3000 余篇。另著有《像黑客一样使用命令行》[1]、《拥抱下一代容器化工具：Podman、Buildah 和 Skopeo》[2]，译有《笨办法学 Git》[3]、《Perl 程序员应该知道的事》等图书。Twitter：https://twitter.com/linuxtoy，Mail：xuxiaodong@pm.me[4]。

[1] https://selfhostedserver.com/usingcli-book

[2] https://selfhostedserver.com/nextcontainer

[3] https://selfhostedserver.com/learngit

[4] mailto:xuxiaodong@pm.me

第一章　准备篇

随着云计算的逐渐流行，创建和管理诸如虚拟服务器、网络、存储、负载均衡等基础设施的方式已然发生转变。传统的手动或者脚本管理方式虽然能够应付数台服务器这种少量使用场景，但是对于多则成百上千的大规模服务器数量来说，显然不再适用和高效。为此，我们需要一种能够自动化的、可重复的以及可靠的创建和管理云基础设施的方法。来自 Google（谷歌）、Amazon（亚马逊）、Netflix（网飞）等互联网公司的做法表明，基础设施即代码正是用来解决此类问题的最佳实践。那么，究竟什么是基础设施即代码呢?

1.1　什么是基础设施即代码

基础设施即代码，其对应的英文为 Infrastructure as code，意指通过编写并执行代码来定义、部署以及更新基础设施。采用基础设施即代码，需要我们转变过去的思维，将云主机、网络（包括路由、防火墙）、存储等所有基础设施都当作软件来对待，从而能够有效利用软件开发的良好实践。使用源代码来表示基础设施，不仅可对其进行编程处理，而且能版本化、共享、重用、调试，甚至在出现错误时还可能还原。对于在软件开发生命周期中进行的代码评审、自动化测试、CI（Continuous integration，持续集成）和 CD（Continuous delivery，持续交付）等活动，也可以同样应用到基础设施上。因此，我们就能创建更加容易追溯、监视和调试的服务运行环境。

1.1.1　基础设施即代码的价值

所有的云计算平台都无一例外的提供了 Web 操作界面。对于大部分人而言，通过 Web 操作界面只需要点点鼠标即可创建基础设施。显然，它看起来非常容

易使用。那么，我们为什么不推荐这种手动管理基础设施的方式呢？这是因为手动管理方法一般具有下列弊端：

- 十分容易出错。仔细回想一下，你上次手动创建基础设施犯错是什么时候？是不是才没过几天？
- 很难创建多个相同且一致的环境。每一个环境都很特殊。它们的配置各不相同。久而久之便成了雪花服务器。这些服务器脆弱不堪，一触即崩。“标准”二字在他们的字典里根本就没有存在过。
- 非常耗时。虽然这是手动操作所需付出的代价之一，但是我们实在没有必要为此浪费大把的时间，毕竟我们还有更好的选择。
- 想要保持环境最新真的是难。给老掉牙的系统打补丁？拜托别碰，你可能会陷入“依赖地狱”。
- 最重要的是你的基础设施无法像收银小票一样长期保留下来。它只散见于几篇过时的文档或某一时刻的大脑中。

而使用基础设施即代码这种实践将带来多方面的好处，我们认为它最重要的价值在于：

- **版本控制**

  我们可以将用来定义基础设施的代码存储在版本控制系统（如 Git）中。一方面，从提交的版本记录我们可追溯基础设施从创建到更新的整个历史过程；另一方面，假如基础设施遇到故障，那么我们便能根据已有的历史记录进行排查。此外，利用版本控制系统，我们甚至还能回滚具有问题的基础设施，从而将其还原到正常状态的版本。

- **自动化**

  执行定义的基础设施代码通常是自动化完成。显然，它比在 Web 界面中通过手动点击按钮来创建基础设施的过程要快。同时，它也更安全，手动操作一般容易导致人为错误；而自动化过程则更加一致且可重复。

- **重用**

  通过把定义基础设施的代码打包成模块，我们从而能够将其变成可以重复使用的组件。对于开发、测试及生产环境中的同类型基础设施，我们无需从头开始构建，只需加以重复利用。因此，重用不仅可以节省我们创建基础设施的时间，而且经过测试的模块也将使基础设施更加可靠。

- **可执行的文档**

  从传统上讲，创建基础设施的过程要么被记录到文档中，要么存在于某个运维专职人员的大脑里。使用文档的问题是，它总是容易过时，而且维护它需要耗费大量的时间。而将特定技术锁定在专职人员身上有时则让情况更加严重，假如该名人员不在工作岗位上，那么将没有别的人知道如何创建基础设施。如果我们将基础设施通过代码的形式定义，那么团队中的每一个人不但都可以查看或评审这些代码，而且还能直接予以执行。

周而复始的手动管理基础设施，时间长了难免不变得枯燥和乏味。如此工作既无创意，也无挑战。通过基础设施即代码的方法在改进我们工作的同时，让我们专注于从事更有价值的事情。

1.1.2 基础设施即代码的工具

我们已经看到了采用基础设施即代码具有巨大的优势。然而，我们如何才能实现它呢？根据 Yevgeniy Brikman 的说法[1]，如果你想要实现基础设施即代码，那么至少包括以下几类工具可以选择使用：

- **脚本**

  通过编写脚本来自动化管理基础设施，这是最直接，同时也是历史最悠久的方式。在早期，用来编写脚本的语言通常以 Shell 为主（如 GNU Bash[2]），但后来更加通用的编程语言（如 Perl[3]、Python[4]、Ruby[5] 等）也加入了进来。使用脚本的好处是简单且上手相对更快，一般可以利用熟悉的编程语言来编写。然而，其缺点也非常明显，除了要编写大量的定制化代码之外，后期的维护更是颇成问题。

- **配置管理工具**

---

[1] https://blog.gruntwork.io/a-comprehensive-guide-to-terraform-b3d32832baca

[2] https://www.gnu.org/software/bash/

[3] https://www.perl.org/

[4] https://www.python.org/

[5] https://www.ruby-lang.org/

当下主流的配置管理工具包括 Ansible[6]、SaltStack[7]、Chef[8]、Puppet[9]、CFEngine[10] 等等，这些工具用来在现有的服务器上自动安装并配置软件。与脚本相比，一方面，使用这类工具通常采用描述或声明式的 DSL（特定领域语言）来编写。由于不再需要学习专门的编程语言，所以在使用难度上大为降低。

另一方面，使用脚本来实现幂等通常较为困难，而通过配置管理工具却能轻易实现。所谓幂等，就是说不管代码执行多少次，其最终行为都将是正确的。例如，假设我们想要为 `/etc/hosts` 追加一行条目，如果使用脚本，那么每次执行可能都将添加相同的内容，最终导致这些内容重复存在于文件中。经验和教训告诉我们，编写仅仅工作的脚本很容易，但要编写好好工作的脚本却十分困难。相反，使用配置管理工具则不会存在这样的问题。这是因为它们会根据描述或声明的状态进行判断，若该文件没有这行条目则添加，反之则什么也不做。

此外，脚本一般在单机上执行，而且临时脚本通常用完即弃。配置管理工具则往往针对分布式执行而设计，能够同时管理大量的远程服务器。

- **服务器模板工具**

  代替在现有的服务器上安装和配置软件，我们也可以直接将预装好的环境打包成服务器模板。这种模式被称为不变基础设施模式，该模式现在已经得到了越来越多的采用。

  一般用来制作服务器模板的技术包括容器化技术和虚拟化技术，而相应的工具则有 Docker[11]、Podman[12]、Packer[13]、Vagrant[14] 等等。通过这些工具打包好的服务器模板称为镜像，将镜像进行部署后最终成为运行的实例。

  不变基础设施模式的惯用做法是，从来不对实例做任何变更。但是，当不得不进行变更时，则从服务器模板创建新的镜像。另外，如果运行的实例

[6] https://www.ansible.com/
[7] https://www.saltstack.com/
[8] https://www.chef.io/
[9] https://puppet.com/
[10] https://cfengine.com/
[11] https://www.docker.com/
[12] https://podman.io/
[13] https://packer.io/
[14] https://www.vagrantup.com/

出现故障，那么也不用对它进行修复，而是将它抛弃，并根据镜像重新部署新的实例。

- **服务器准备工具**

  无论是配置管理工具，还是服务器模板工具，它们定义的代码都针对现有的服务器运行。与它们不同的是，服务器准备工具则用来创建服务器本身。这类工具包括 Terraform[15]、AWS CloudFormation[16]、OpenStack Heat[17]等等。

本书将专注于介绍 Terraform，根据我们长期观察，Terraform 是目前用来编写基础设施即代码的最佳开源工具之一。我们喜爱 Terraform，不仅因为它支持 Amazon Web Services（AWS）[18]、Google Cloud[19]、Microsoft Azure[20]、DigitalOcean[21]、Linode[22]、Vultr[23]、阿里云[24]、腾讯云[25]等市场上广泛的云供应商，而且也在于它的生态越来越成熟。

1.2 Terraform 简介及工作原理

与 Vagrant 和 Packer 一样，Terraform 也是 HashiCorp[26] 旗下的开源产品。Terraform 使用 Go[27] 编程语言开发而成，于 2014 年 7 月 28 日推出了第一个公开发布版本。从发布至今的 5 年间，Terraform 已经得到了包括 AWS、IBM Cloud[28]、Google Cloud、Microsoft Azure、Oracle Cloud[29]、阿里云、腾

[15] https://www.terraform.io/
[16] https://aws.amazon.com/cloudformation/
[17] https://www.openstack.org/software/releases/stein/components/heat
[18] https://aws.amazon.com/
[19] https://cloud.google.com/
[20] https://azure.microsoft.com/
[21] https://www.digitalocean.com/
[22] https://www.linode.com/
[23] https://www.vultr.com/
[24] https://www.alibabacloud.com/zh
[25] https://cloud.tencent.com/
[26] https://www.hashicorp.com/
[27] https://golang.org/
[28] https://www.ibm.com/cloud
[29] https://www.oracle.com/cloud/

讯云、VMware vSphere[30]、OpenStack[31] 等多种公有云及私有云的支持。因为 Terraform 在目前是如此流行，掌握它的用法自然成为了一项必备的 IT（Information technology，信息技术）技能。

1.2.1 Terraform 简介

Terraform 是用来创建和管理基础设施的得力而优雅的工具。它通过自创的 HCL[32]（HashiCorp Configuration Language，HashiCorp 配置语言）配置语言来声明想要创建的基础设施的最终状态。例如，假设我们打算在 AWS 上创建一台计算实例，那么只需为其指定实例的类型（由此决定可以使用多少 CPU、内存及网络带宽）和操作系统的镜像（如 Debian GNU/Linux）即可，而完全不必在意该计算实例是如何创建的过程。因为，Terraform 将为我们关心这一切。

在应用声明配置之前，Terraform 做得非常好的一点是，它允许我们先执行计划，故而能够让我们清晰的看到即将实施的基础设施的具体变化。通过执行计划，我们不仅可以详细了解创建、变更以及销毁的是哪些基础设施，而且还能据此判断是否需要最终加以应用配置。

Terraform 还能与 Ansible、SaltStack、Chef、Puppet 等配置管理工具集成，在创建基础设施的同时，执行安装、配置软件及服务。利用 Terraform 这项功能，我们可以重用现有的配置管理资源，从而将以往的工作流和 Terraform 整合在一起。

目前，Terraform 主要用于管理云计算平台和 SaaS（Software as a Service，软件即服务）基础设施，其范围包括虚拟服务器、数据存储、网络、DNS、CDN、数据库、监视服务等等。

1.2.2 Terraform 的工作原理

Terraform 的工作原理如图 1.1 所示[33]。以自顶向下的视角来看，首先，我们通过人类可读的 HCL 配置语言将基础设施表示为代码。在代码中，我们声明想

[30] https://www.vmware.com/products/vsphere.html

[31] https://www.openstack.org/

[32] https://github.com/hashicorp/hcl

[33] https://www.hashicorp.com/products/terraform/

要创建的基础设施的最终状态。这一步，我们既可以从 Terraform 公开的模块仓库及社区中重用基础设施资源，也能够将 Terraform 状态文件存储到云中或从云中同步该文件以便与整个团队的所有成员协作。

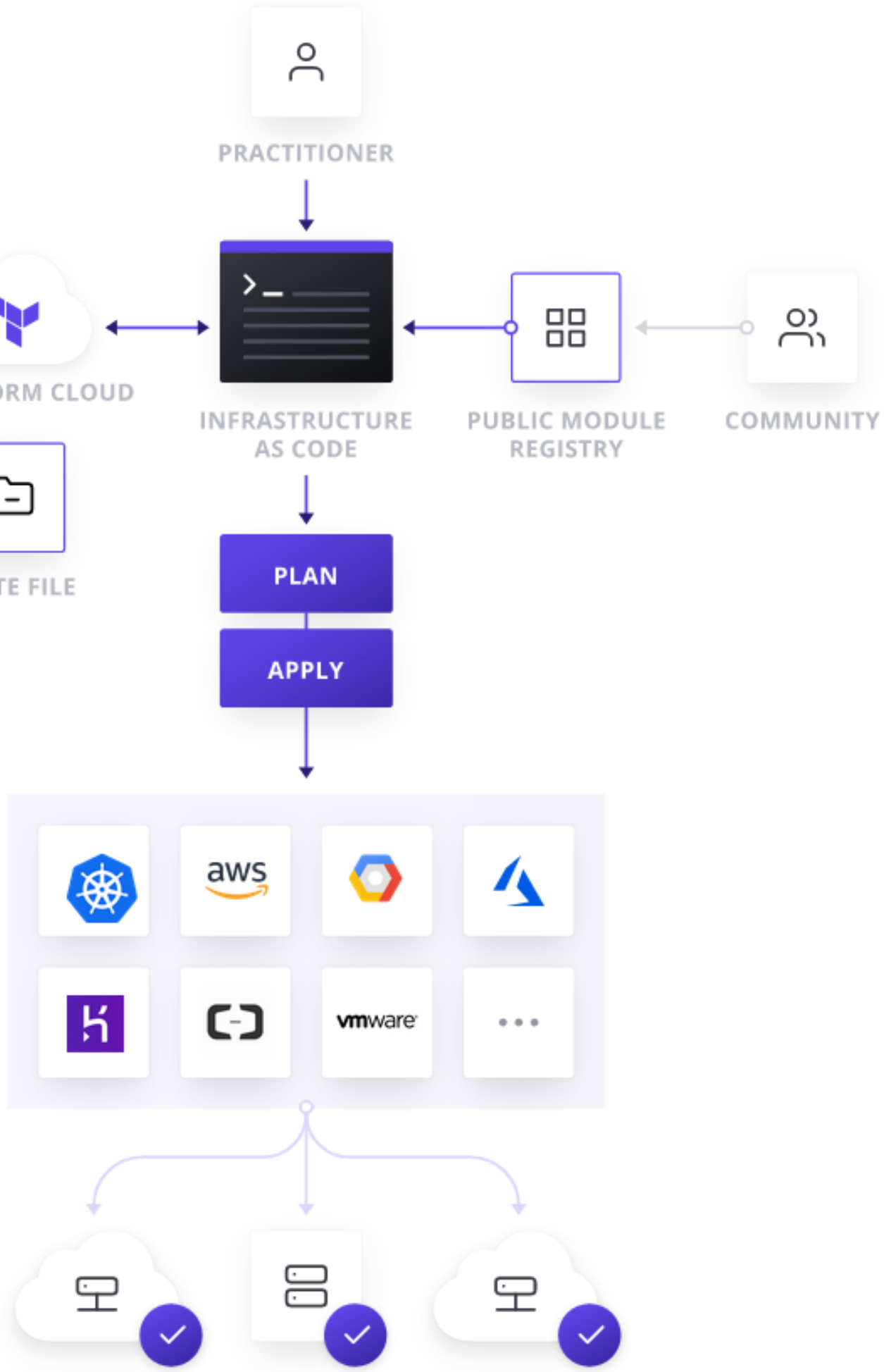

图 1.1: Terraform 的工作原理

接着，Terraform 负责解析我们在上一步编写的基础设施代码，并生成相应的执行计划。这份执行计划向我们展示 Terraform 实际上在应用代码时所做的具体操作。

最后，Terraform 应用我们定义的基础设施代码。在内部，Terraform 将 HCL 配置语言转换成向云供应商（如 AWS、Google Cloud、Microsoft Azure、阿里云等）或应用服务（如 Kubernetes、Heroku 等）进行调用的 API（Application programming interface，应用程序编程接口），从而最终实现创建和管理基础设施。

1.3 Terraform 的使用场景

我们选择 Terraform 的缘由之一在于它具有广泛地使用场景，这主要体现在下列方面：

- **自助式服务**

  在一个团队中，通常由运维人员负责准备基础设施，并执行相应的管理过程。随着业务规模的增长，数量有限的运维人员很容易成为瓶颈。利用 Terraform，运维人员能够将基础设施定义成标准的资源和技术栈，从而以自助式的服务提供给团队中的其他人员消费。

- **多层应用程序**

  Terraform 聚焦于基础设施的部署，其范围涵盖计算实例、存储及网络。这使它成为构建多层应用程序的理想工具。无论是具有数据库后端的经典 Web 应用，还是具有 Web、缓存、中间件、数据库的多层应用，每一层都可以通过 Terraform 定义成代码。代码一旦在手，它们将更容易管理和自动伸缩。

- **一致的环境**

  在运维工作中一个常见的现实问题是，开发环境和测试环境总是很难与生产环境保持一致。结果即便应用程序通过了测试，然而在生产环境中仍有可能无法正常运行。通过 Terraform，我们把生产环境的基础设施代码化，然后将其与开发环境和测试环境共享，从而确保所有环境的一致性和兼容性。

- **CD（持续交付）**

  如果你的应用程序已经通过 CI（持续集成）实现了自动化构建，那么不妨考虑更进一步，将 CI 中构建的制品进行自动化部署。使用 Terraform 自动准备各种环境，从而实现更有针对性的按需部署。同时，把它们串联在一起，形成真正的自动化部署流水线。

- **管理基础服务**

  除了管理各种云基础设施之外，Terraform 也能管理许多基础服务和工具。例如，GitHub[34] 组织和仓库、GitLab[35] 群组与项目、Grafana[36] 数据源及 Dashboard 等等。不管你信不信，有人甚至使用 Terraform 来订购匹萨[37]。Terraform 的用途之广由此可见一斑。

1.4 安装 Terraform

Terraform 支持 Linux、macOS、Windows、FreeBSD、OpenBSD、Solaris 等多种操作系统，因其使用单个二进制包进行分发，所以只需下载后将 Zip 文件解包，并移到操作系统的 **PATH** 所在目录中即可完成安装。

1.4.1 在 Linux 上安装 Terraform

如果你想要在 Linux 上安装 Terraform，那么在打开 Terraform 的官方下载页面[38]之后，找到适合 Linux 架构的版本（本书将使用 64 位架构，Terraform 也具有 32 位架构和 ARM 架构）。截至编写本书时为止，Terraform 的最新版本为 0.12.9。接着，按照下列步骤执行操作：

1. 使用 `wget`（或 `curl`）命令下载 Terraform 的 Zip 压缩包：

```
mkdir ~/terraform
cd ~/terraform
```

[34] https://github.com/
[35] https://about.gitlab.com/
[36] https://grafana.com/
[37] https://github.com/ndmckinley/terraform-provider-dominos
[38] https://www.terraform.io/downloads.html

```
wget https://releases.hashicorp.com/terraform/0.12.9/
terraform_0.12.9_linux_amd64.zip
```

2. 执行 `unzip` 命令来解包：

```
unzip terraform_0.12.9_linux_amd64.zip
```

3. 将 `terraform` 二进制文件移到操作系统的 **PATH** 所在目录 `/usr/local/bin`：

```
sudo mv terraform /usr/local/bin
```

最后，我们通过执行 `terraform version` 命令来验证 Terraform 是否安装成功：

```
terraform version
Terraform v0.12.9
```

`terraform version` 命令的输出结果说明我们正在使用 Terraform 的 0.12.9 版本，同时这也意味着 Terraform 的安装一切正常。

值得一提的是，通常使用 Linux 发行版所带的包管理器（如 `apt`、`yum`、`pacman` 等等）也可以安装 Terraform。不过，极有可能不是最新的 Terraform 版本，各位读者朋友请在仔细斟酌后选择使用。

1.4.2 在 Windows 上安装 Terraform

除了与 Linux 类似的手动安装 Terraform 方式外，我们也可以通过 Chocolatey[39] 这个包管理器将 Terraform 安装到 Windows 上。如果你的 Windows 操作系统中还没有 Chocolatey 的话，那么不妨先按它的介绍文档[40]进行安装。Chocolatey 一旦安装完成之后，就能利用 `choco` 命令来安装 Terraform：

[39] https://chocolatey.org/

[40] https://chocolatey.org/docs/installation

```
choco install terraform
```

1.4.3 在 macOS 上安装 Terraform

在 macOS 上，我们同样可以按照 Linux 的方式来手动安装 Terraform。不过，比它更方便的安装方法是使用 Homebrew[41] 包管理器：

```
brew install terraform
```

1.4.4 利用 tfenv 管理多个 Terraform 版本

tfenv[42] 是受 rbenv[43] 启发编写而成的实用工具，利用它我们可以非常方便的管理操作系统中的多个 Terraform 版本。对于应用多版本的 Terraform 场景而言，tfenv 显得尤其有用。例如，在生产环境中我们使用 Terraform 的稳定版本来创建和管理基础设施。同时，我们使用 Terraform 的测试版本在其它环境试验新增特性。

目前，tfenv 支持 Linux、macOS 以及 Windows 操作系统，包括 64 位和 ARM 架构。

1.4.4.1 安装 tfenv

以 Linux 和 macOS 操作系统为例，在安装 tfenv 之前，确保系统中具有 Git 源代码管理工具。然后，执行以下命令将 tfenv 从 GitHub 上克隆到本机的 `~/.tfenv` 目录，并对 tfenv 的可执行文件建立相应的符号链接：

```
git clone https://github.com/tfutils/tfenv.git ~/.tfenv
sudo ln -s ~/.tfenv/bin/* /usr/local/bin
```

当上述命令执行完成后，接着运行 `tfenv` 命令以便验证其安装确实可用：

---

[41] https://brew.sh/

[42] https://github.com/tfutils/tfenv

[43] https://github.com/rbenv/rbenv

```
tfenv
tfenv 1.0.1-8-gc3f5d5e
Usage: tfenv <command> [<options>]

Commands:
   install       Install a specific version of Terraform
   use           Switch a version to use
   uninstall     Uninstall a specific version of Terraform
   list          List all installed versions
   list-remote   List all installable versions
```

从该命令的输出信息中，我们既可了解 tfenv 的基本用法（包括安装、列出、切换、卸载等操作），也能查看其版本号（1.0.1）。

1.4.4.2 利用 tfenv 安装 Terraform

首先，我们来看看 tfenv 支持安装哪些 Terraform 版本（执行 `tfenv list-remote` 命令）：

```
tfenv list-remote
0.12.9
0.12.8
0.12.7
...
0.12.0
0.12.0-rc1
0.12.0-beta2
0.12.0-beta1
...
0.2.0
0.1.1
0.1.0
```

从最早的 0.1.0 版本到最新的 0.12.9 版本，从 Beta 测试版本到 RC 候选版本，

tfenv 对 Terraform 各个版本的支持真可谓一应俱全。

接着，作为试验，我们安装 Terraform 0.11.14 版本。为此，执行以下命令：

```
tfenv install 0.11.14
[INFO] Installing Terraform v0.11.14
[INFO] Downloading release tarball from https://releases.hashicorp
.com/terraform/0.11.14/terraform_0.11.14_linux_amd64.zip
######################################################################100.0%
[INFO] Downloading SHA hash file from https://releases.hashicorp
.com/terraform/0.11.14/terraform_0.11.14_SHA256SUMS
  INFO Starting keybase.service.
tfenv: tfenv-install: [WARN] Unable to verify OpenPGP signature
unless logged into keybase and following hashicorp
Archive:  tfenv_download.g3jsEy/terraform_0.11.14_linux_amd64.zip
  inflating: /home/xiaodong/.tfenv/versions/0.11.14/terraform
[INFO] Installation of terraform v0.11.14 successful
[INFO] Switching to v0.11.14
[INFO] Switching completed
```

同时，我们也安装一个 Terraform 的最新版本。代替在 `install` 命令后面指定具体的版本号，我们直接使用 `latest` 参数：

```
tfenv install latest
```

在安装完毕后，tfenv 将自动切换到安装的版本。执行 `tfenv list` 命令可以查看当前系统中已经安装的所有 Terraform 版本：

```
tfenv list
* 0.12.9 (set by /home/xiaodong/.tfenv/version)
  0.11.14
```

其中，列表条目前面的 `*`（星号）表示当前正在使用的 Terraform 版本。本例中为 Terraform 0.12.9。

1.4.4.3 切换 Terraform 版本

假如你想要切换系统中已经安装的 Terraform 版本，那么可以使用 tfenv 的 `use` 命令，例如：

```
tfenv use 0.11.14
[INFO] Switching to v0.11.14
[INFO] Switching completed
```

在此，我们将 Terraform 从最新版本（0.12.9）切换到 0.11.14 版本。

再譬如：

```
tfenv use latest
[INFO] Switching to v0.12.9
[INFO] Switching completed
```

这次又还原到最新的 Terraform 版本 0.12.9。

需要指出的是，利用 `tfenv use` 命令切换 Terraform 的版本，其影响范围是全局性的。另一种十分有用的固定 Terraform 版本的方法是使用 `.terraform-version` 文件。假如我们的 Terraform 项目针对的是 0.11.14 系列版本，那么只需在该项目的工作目录执行以下命令即可将 Terraform 的版本设置为 0.11.14。而一旦离开该项目的工作目录，Terraform 仍旧使用全局性的版本。

```
echo 0.11.14 > .terraform-version
terraform version
Terraform v0.11.14

cd ..
terraform version
Terraform v0.12.9
```

1.4.4.4 卸载 Terraform 版本

如果确认先前安装的某个 Terraform 版本不再需要使用，那么可以将其卸载，以便腾出占用的磁盘空间。例如，下列命令将卸载 Terraform 0.11.0 版本：

```
tfenv uninstall 0.11.0
[INFO] Uninstall Terraform v0.11.0
[INFO] Terraform v0.11.0 is successfully uninstalled
```

1.5 配置 Terraform 开发环境

虽然你可以使用任意的文本编辑器来编写 Terraform 代码，但是对 Terraform 支持更佳的文本编辑器将让你用起来更加得心应手，而且生产效率也会更高。在令人眼花缭乱的代码编辑器中，我们确实更偏爱 Visual Studio Code[44]（简称 VS Code）一些。

VS Code 是由微软推出的一款代码编辑器，它不仅开源，而且跨平台，能够在包括 Linux、macOS、Windows 等在内的操作系统上运行。这意味着，当你转换开发系统平台时，原有的开发环境依然可以保持不变。

VS Code 具有名为 IntelliSense 的智能感知系统，除了支持常见的语法高亮之外，IntelliSense 还提供更加智能的自动补全特性，它能够基于变量类型、函数定义以及导入模块进行自动补全。利用智能感知系统的帮助，将极大的提高我们的代码输入效率。

在 VS Code 中所包含的另一项好特性是内建 Git 命令支持。Git 是目前相当流行的分布式版本控制系统，无需额外的 Git 工具，我们直接就可以在 VS Code 中执行诸如暂存文件、比较差异、创建提交等各种 Git 操作，使用非常方便。

此外，VS Code 还包括十分强大的扩展和定制化能力。通过扩展，我们能够根据自身的想法和需求，实现 VS Code 原本所不具有的功能。顺便提一句，对于我们接下来将要介绍的 Terraform 支持即是利用 VS Code 的扩展而实现。

[44] https://code.visualstudio.com/

1.5.1 Terraform 扩展简介

VS Code 的 Terraform 扩展[45]由 Mikael Olenfalk 所开发，该扩展主要为 Terraform 代码带来了下列功能：

- 针对 `.tf`、`.tfvars` 及 `.hcl` 等文件格式进行语法高亮；
- 自动补全 Terraform 资源类型和属性；
- 利用 `terraform fmt` 命令自动格式化 Terraform 代码；
- 通过 TFLint[46] 来 Lint 代码；
- 验证 Terraform 项目；
- 浏览文档符号。

1.5.2 安装 Terraform 扩展

在打开 VS Code 后，按 **Ctrl + p** 组合键启动 **Quick Open**（快速打开）。接着，将以下命令粘贴到文本框中并按**回车键**，如图 1.2 所示：

```
ext install mauve.terraform
```

一旦安装完成，我们就可以在 VS Code 中使用 Terraform 扩展了。

[45] https://marketplace.visualstudio.com/items?itemName=mauve.terraform

[46] https://github.com/wata727/tflint

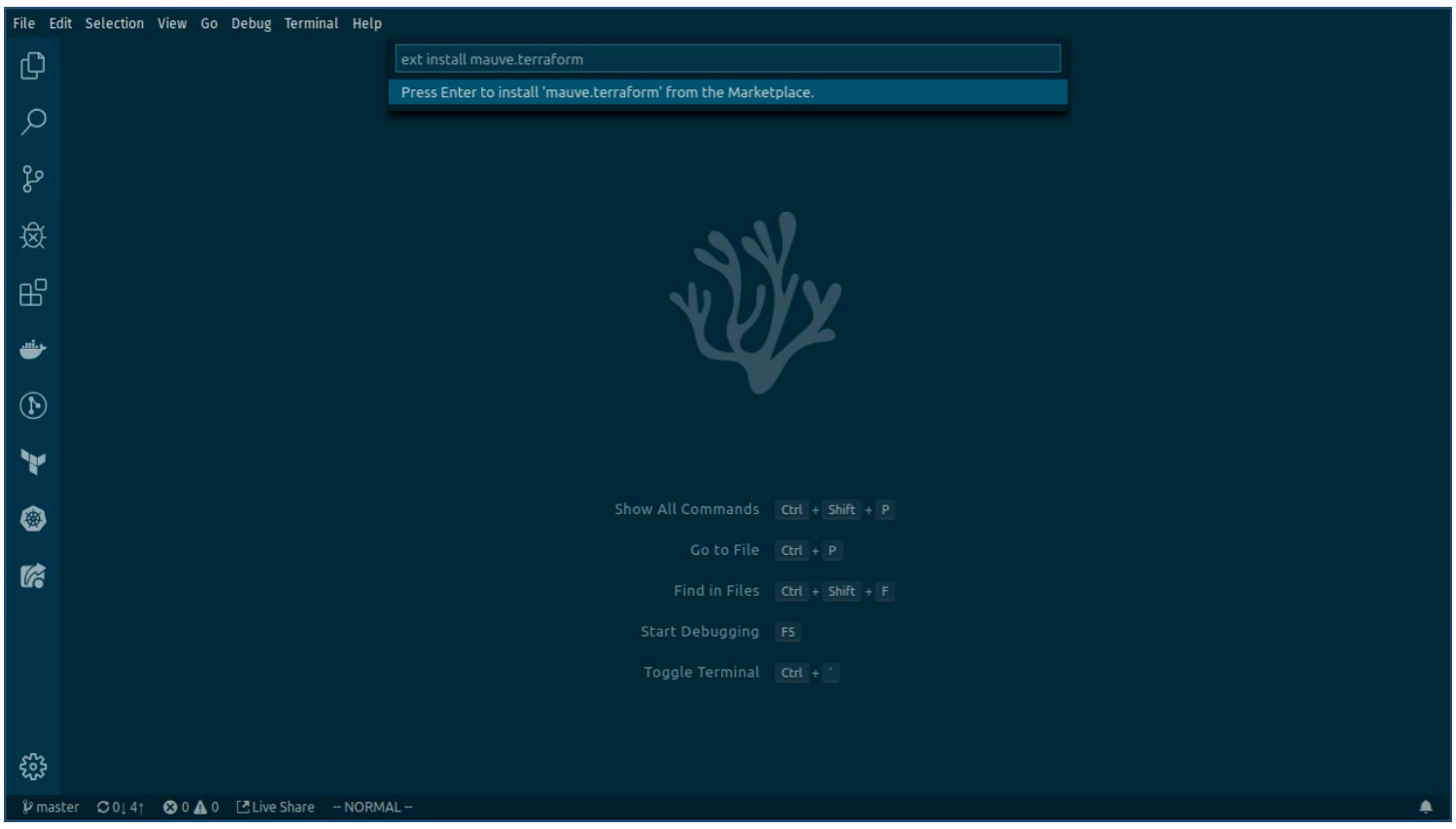

图 1.2: 在 VS Code 中安装 Terraform 扩展